## **Search Engine System Documentation**

YU, Yingxuan (20945139) of 2025S COMP4321 Project Group 16

### **1. Overall Design of the System**

The search engine is a modular system comprising four tightly integrated components designed for efficiency, scalability, and user-centric functionality:

#### **Web Crawler (spider.py)**

* **Breadth-First Search (BFS) Traversal**: Initiates crawling from a user-specified seed URL, systematically exploring linked pages while adhering to a configurable maximum page limit.
* **Content Extraction**: Parses HTML to extract titles, text content, hyperlinks, and metadata (e.g. “Last-Modified” headers, page size).
* **Dynamic Refresh Logic**: Validates page freshness using HTTP “HEAD” requests and conditional checks against cached timestamps.
* **Persistence Layer**: Stores crawled data in webpages.db using SQLite. Sensitive fields (URLs, titles) are base64-encoded to ensure compatibility with special characters.

#### **1.2 Indexer (indexer.py)**

* **Inverted Index Construction**: Generates two indices:
  + **Body Index**: Processes page body text, tokenizing keywords and phrases (2–5 words) after stop word removal and Porter stemming.
  + **Title Index**: Prioritizes title content with boosted weights, applying the same tokenization pipeline.
* **TF-IDF Weighting**: Computes term frequency (TF) and inverse document frequency (IDF) to rank keyword relevance.
* **Database Optimization**: Splits indices into body\_inverted\_index.db and title\_inverted\_index.db to accelerate query processing.

#### **1.3 Retrieval Function (retrieval.py)**

* **Query Parsing**: Supports phrase queries (e.g. "Hong Kong") and standard keyword searches.
* **Vector Space Model**: Represents documents and queries as TF-IDF vectors, calculating relevance via cosine similarity.
* **Score Boosting**: Applies multipliers for title matches (2×) and phrase occurrences (1.5× for body, 2× for titles).
* **Dynamic Crawling**: Fetches uncached pages on demand to ensure up-to-date results.

#### **1.4 Web Interface (webui.py)**

* **Flask-Based UI**: Renders a minimalist interface with real-time search capabilities.
* **Input Sanitization**: Automatically filters non-ASCII characters and converts smart quotes to standard equivalents.
* **Visual Enhancements**: Highlights visited links, displays keyword summaries, and formats results with metadata (e.g. page size, last-modified dates).

### **2. File Structures in the Index Database**

#### **2.1 webpages.db Schema**

* **Table**: **“webpages”**
  + “url” (Base64): Primary key; ensures URL uniqueness and safe storage.
  + “title” (Base64): Page title encoded to handle multilingual text.
  + “date” (Base64): ISO-formatted timestamp of the last page modification.
  + “size” (Base64): Page size in bytes.
  + “body\_keywords”: Serialized dictionary of “{keyword:frequency}” pairs, where keyword and frequency are all base64.
  + “parent\_links”/“child\_links”: Comma-separated lists of base64-encoded URLs.
  + “is\_start” (Base64): Flag (“0”/“1”) indicating whether the URL is the seed.

#### **2.2 Inverted Index Databases**

* **Body Index (body\_inverted\_index.db)**
  + **Table**: **inverted\_index**
    - “keyword” (Base64): Stemmed term or phrase (e.g. "machine learn" for "machine learning").
    - “postings” (Text): Encoded list of “url:tf:tfidf” entries, where:
      * “url”: Base64-encoded document URL.
      * “tf”: Base64-encoded term frequency in the document.
      * “tfidf”: Base64-encoded TF-IDF score rounded to 4 decimal places.
* **Title Index (title\_inverted\_index.db)**
  + Identical structure to the body index but with terms extracted from page titles.

### **3. Key Algorithms**

#### **3.1 BFS Crawling with Conditional Refresh**

* **Queue Management**: Uses a “deque” to manage unvisited URLs, ensuring FIFO processing.
* **Page Update Detection**:

1. Sends “HEAD” requests to check “Last-Modified” headers.
2. Compares timestamps with cached values; re-crawls stale pages.
3. Prunes orphaned pages (those with no active parent links).

#### **3.2 Tokenization and Stemming Pipeline**

* **Stopword Removal**: Filters common words (e.g. "the", "and") using a predefined list (stopwords.txt)
* **Phrase Extraction**: Identifies 2–5-word sequences from raw text (e.g. "search engine optimization").
* **Porter Stemming**: Reduces words to root forms (e.g. "running" → "run") for consistent indexing.

#### **3.3 TF-IDF Calculation**

* **Term Frequency (TF)**:
* **Inverse Document Frequency (IDF)**:
* **TF-IDF Weight**:

, normalized by the maximum tf in the document

#### **3.4 Query Processing and Ranking**

* **Query Vectorization**:
  + Tokenizes and stems query terms.
  + Computes TF-IDF weights using document frequencies from both indices.
* **Cosine Similarity**:
* **Score Adjustment**:
  + Title matches: score \*= 2
  + Body phrase matches: score \*= 1.5
  + Title phrase matches: score \*= 3

### **4. Installation and Deployment**

#### **4.1 Prerequisites**

* **Python 3.13**: Required for async features and library compatibility.
* **Dependencies**:
  + Windows: pip install -r requirements.txt
  + Arch Linux: sudo pacman -S python python-requests python-lxml nltk-data python-nltk python-flask

#### **4.2 Execution**

* **Start the Server**: python webui.py
* **Access the UI**: Navigate to <http://localhost:11451> in a browser
* **Note: It may be very slow on the first query. This is normal, because our search engine starts to crawl the web pages on the first query, not on start.**

#### **4.3 Customization**

* **Adjust Crawling Limits**: Modify “Max Crawled Page” in the UI to control resource usage.
* **Adjust Result Limits**: Modify “Max Results” in the UI to control resource usage.
* **Update Stopwords**: Edit stopwords.txt to include domain-specific noise words.

### **5. Advanced Features Beyond Requirements**

#### **5.1 Phrase Query Support**

* Users can enclose multi-word phrases in quotes (e.g. "deep learning") for exact matches.
* The engine prioritizes documents containing these phrases in titles or body text.

#### **5.2 Dynamic Link Visualization**

* Search results display parent/child links as hyperlinks, enabling users to navigate the crawled graph.

#### **5.3 Keyword Summarization**

* Each result shows a “Keywords” field listing the top 5 stemmed terms and their frequencies (e.g. algorithm 12; data 9; ...).

#### **5.4 Input Sanitization and Compatibility**

* Automatically converts non-ASCII characters and smart quotes to standard equivalents.
* Validates URLs to handle typos (e.g. prepending “http://” if missing).

#### **5.5 Session-Based Link Tracking**

* Visited links are highlighted in purple during a session, stored client-side via “sessionStorage”.

### **6. Testing and Validation**

Submit a query without non-ASCII characters (e.g. “Café”)

* **Expected**: Valid results returned.
* **Result**: Passed.

### **7. System Evaluation**

#### **7.1 Strengths**

* **Modularity**: Components are decoupled, enabling independent updates (e.g. replacing the crawler with a distributed alternative).
* **Efficiency**: SQLite indexing and in-memory vector operations ensure sub-second response times for small-to-medium corpora.
* **Robustness**: Gracefully handles malformed HTML, encoding errors, and network timeouts.

#### **7.2 Weaknesses**

* **Scalability**: In-memory vector calculations become impractical for corpora exceeding 10,000 documents.
* **Language Support**: Limited to English due to hardcoded stopwords and stemming rules.
* **Concurrency**: Single-threaded crawling/indexing limits performance on multi-core systems.

#### **7.3 Design Trade-Offs**

* **Simplicity vs. Performance**: Chose SQLite over Elasticsearch/PostgreSQL to reduce setup complexity, sacrificing horizontal scalability.
* **Accuracy vs. Speed**: Prioritized exact phrase matching over fuzzy search to maintain precision, increasing query latency.

#### **7.4 Future Improvements**

* **Distributed Architecture**: Implement Apache Spark or Scrapy for large-scale crawling/indexing.
* **Relevance Tuning**: Integrate PageRank or BERT-based embeddings for semantic similarity.
* **Multilingual Support**: Add language detection and locale-specific tokenization.

#### **7.5 Feature Roadmap**

* **Autocomplete**: Suggest queries using Trie-based prefix matching.
* **Snippet Generation**: Display contextual text around keyword hits.
* **User Feedback Loop**: Allow votes to improve ranking (e.g. “Was this result helpful?”).

### **8. Contribution**

* YU, Yingxuan ([xuangeyouneihan](https://github.com/xuangeyouneihan), me):

75%. I wrote almost all the code in the project and wrote all the documentation after Phase 1.

* DU, Maosen ([ThisIsNotCodingJellyfish](https://github.com/ThisIsNotCodingJellyfish)):

24%. He wrote the documentation before Phase 1 submission, but he left from study after that.

* WU, Lixin:

1%. He applied late drop for this course, and gave DU, Maosen and me some ideas.

* LIN, Xuanyu:

0%. He did not even contact us three.